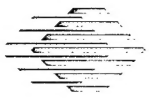
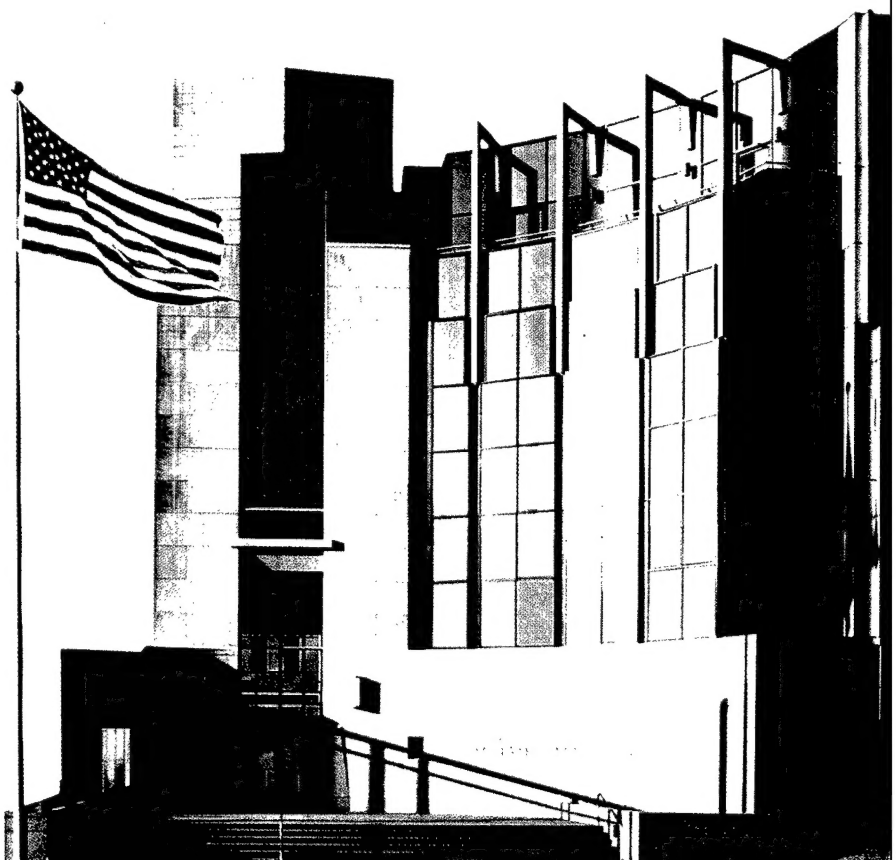


Carnegie Mellon
Software Engineering Institute

Guidance on Commercial-Based and Open Systems for Program Managers

Patrick R.H. Place

April 2001



SPECIAL REPORT
CMU/SEI-2001-SR-008

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Guidance on Commercial-Based and Open Systems for Program Managers

CMU/SEI-2001-SR-008

Patrick R. H. Place

April 2001

COTS-Based Systems Initiative

Table of Contents

List of Figures

Acknowledgements

This report is based on a number of other documents. I wish to acknowledge the authors of the works listed below.

1. "Commercial Item Acquisition: – Considerations and Lessons Learned." C. Albert and E.J. Morris.
   http://web1.deskbook.osd.mil/reflib/DDOD/005EO/005EODOC.HTM#T2

2. "COTS-Based Systems for Program Managers (briefing)." L. Brownsword, P. Oberndorf, and C. Sledge. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1999.
   http://www.sei.cmu.edu/cbs/course_desc/CBSforPMsBriefDisc.html - size 20.0K

3. "ESC Program Manager's Handbook on Use of Commercial Software." D. Carney, P. Oberndorf, E.J. Morris, J. Clapp, P. Engert, and S. Meehan. December 1998.

4. "COTS and Open Systems." P. Oberndorf. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. February 1998.

I also wish to thank John Foreman and Ed Morris for their insightful comments on this document.

Executive Summary

There is increasing pressure for program managers to use commercial items within the systems they are acquiring. At the same time, there is a desire that they also adopt an open systems strategy. Use of both commercial items and open systems changes the nature of development and acquisition; both approaches provide constraints, outside the program managers' control, to which the program must adapt.

Use of either a commercial item or open systems strategy raises issues that a traditional government development wouldn't face. In both strategies, the program manager gives up some control, either to the marketplace or to the bodies defining the open systems standards. In some cases, the two strategies will conflict and a program manager must choose between them. Although many of the issues arise in the technical arena and, as such, are the purview of the contractor, the program manager must be aware of the issues and the effect they have on the program both in the short and long term. Without such awareness, the program manager will be unable to react appropriately should the strategies cause unpredicted delays to the program.

This document discusses some realities of using commercial items; guidance is provided in

- managing the gap between system requirements and the capabilities of commercial items
- the changing role of evaluation
- the additional skills needed by a program manager
- the changing nature of inspection and testing
- the importance of a strong relationship with the vendor
- the need to start considering maintenance issues earlier in the life cycle
- the increased complexity when multiple commercial items are employed

Similarly, the realities of open systems are discussed and guidance is provided in

- the selection of appropriate open system standards
- the requirement for new capabilities in conformance testing
- the changes in schedules and costs
- the conflicts between commercial item and open systems strategies

The purpose of this guidance is not to dissuade the program manager from adopting either a commercial item or an open systems strategy – such strategies are inevitable and desirable in today's environment – but, rather to inform the program manager of approaches to coping with the changes these strategies bring.

Abstract

With increasing frequency, program managers are acquiring systems based on both commercial products and open systems. This brings both benefits and risks. Although the benefits of using commercial products and open systems are widely publicized, the risks are often overlooked. This document discusses various risks and provides guidance that may be used to mitigate those risks.

1 Introduction

This guidance contained herewith is intended for program and assistant program managers who will acquire systems containing commercial software products while adopting an open system strategy. The purpose of this guidance is to anticipate questions that may arise as program managers routinely deal with commercial software and open systems and to provide practical advice on how to deal with the risks that arise from a commercial-software-based acquisition.

1.1 Terms and Definitions

A number of imprecisely defined terms are used throughout the software community. The use of these terms can lead to misunderstandings, precisely because of the different meanings. We will define the most important terms used within this document. However, we must also caution that the definitions are intentionally loose; they are defined well enough that a common understanding may be shared, but not so rigidly that they form a straightjacket for our thinking.

COTS and Commercial

Beyond the expansion of COTS to "commercial off-the-shelf," there is little agreement on the meaning of COTS and, hence, consistency in the way it is used. Further, although the acronym "COTS" appears in some government documents and in many discussions, it is used inconsistently. The Federal Acquisition Regulations [FAR] provide extensive definitions of many similar items (e.g., "commercial item," "non-developmental item") but make no mention of the term "COTS."

The most common problem stems from the word "commercial" – for some this means that an item is available for sale while for others this means that the item has been sold (whether widely or at all is simply a matter of degree). Confusion can also arise based on whether or not an item is really "off-the-shelf." This is particularly true for government acquisitions where some of the commercial items (e.g., radar systems) are of limited popular appeal and therefore where a vendor may be more inclined to modify the product to fit the government needs. In cases of modifications to COTS products, the question arises – to what degree should the product be considered commercial or "off-the-shelf?"

Arguing precise definitions of what is and what is not COTS, though, is somewhat fruitless as many of the issues faced by a program manager will be similar (though the effects of these COTS issues may be increased or decreased depending on where on the "off-the-shelf" scale

the product lies). Therefore, for this document, we intend that commercial items should be the most inclusive definition possible, regardless of whether or not the items are actually "off-the-shelf."

Thus, our definition of *commercial items* includes all items available for sale, whether they are of mass appeal (e.g., operating systems) or of a limited, government-oriented, appeal, as are radar systems. Our definition includes items that are available for use "as-is" as well as those that may undergo some tailoring before being deemed suitable for government use. We will, for the most part, use the term "commercial"; however, we will also use the term "COTS" when we refer to an item that is really "off-the-shelf."

Commercially Based Systems

Just as there is confusion over the term "COTS," there is also no single definition of "commercially based system." This is, in part, due to there being many systems that are "commercially based." We define a spectrum of "commercially based systems," the spectrum being based on the ways the systems incorporate commercial items. Two extremes of the spectrum are defined by

- A *commercial-solution* system, in which a single product, or small suite of products provides the essence of the solution in the application area. This type of system is generally available from a single vendor (though it may also consist of other vendors' products), and is constructed by tailoring the product (or suite of products) to specific business needs. Examples of this type of system exist in personnel management, financial management, manufacturing engineering, and so on.
- A *commercial-intensive* system, in which the system capabilities are achieved through the integration of several commercial items from several vendors. In such cases, either the application area is unique, or the scope of the system exceeds the capabilities of products found in the commercial marketplace and precludes a single commercial solution.

These two are extremes, and most systems fall between them. Given the broad range of possibilities it is likely that no single collection of rules applies. Different types of systems require an emphasis on different kinds of engineering activities. Understanding these activities is fundamental to managing a system's implementation and sustainment. Thus, one of the first steps in planning a program is determining where in the array of possibilities your system falls.

Contractor, Integrator, Vendor

The roles of contractor, integrator, and vendor are related but also conceptually distinct. Even though the roles of "contractor" and "integrator" are often performed by the same entity, we can distinguish between them. The contractor is the entity under contract to the government and from whom the program manager expects to receive a delivered system. The integrator is the entity that performs the task of integrating the different commercial items into a func-

tioning system. A vendor is a company that sells products in the marketplace. Of course, it is also possible that this role may be combined with the others. The contractor (or integrator) may also be the vendor of several of the commercial items included in the system.

Open Systems

An open system is something of a misnomer. In reality, we frequently use the term "open system" to refer to some component of our system where the interface to that component is fully defined, is available to the public, and is maintained according to group consensus. The essence of such systems is that it is possible to find multiple vendors selling products that adhere to the same interface standard. It should be noted, though, that this doesn't automatically mean that products from different vendors are easily interchanged.

An open system strategy attempts to maximize use of open system components. It is likely that the adoption of an open system strategy will not lead to an "open system" but to a system composed of open system components.

1.2 Goal of this Guidebook

This guidebook is based in practicality: as program managers implement the acquisition reform policies that encourage and affect use of commercial items and open systems, they will face complex issues stemming from the use of these items.

Commercial items aren't used in a vacuum but, rather, in the context of a system, and this raises associated issues and concerns – upgrade and maintenance, interface standards, requirements management, and so on. There are also programmatic issues (e.g., the nature of the funding needed for certain activities, the division of an acquisition into a number of phases, the drive toward shorter deployment cycles) that must be reconciled with the frequently changing nature of the commercial marketplace. Finally, there are some issues that are not, in themselves, specific to commercial items but that take on increased significance when such items play an important role in an acquisition.

Just as the use of commercial items gives rise to a number of issues, so does the choice of adopting an open systems strategy – not least because adherence to a strict open system policy may conflict with the COTS strategy (many COTS products frequently fail the open system test). Further, as we will see, although an open system approach leads to more stability of system components, it also leads away from "cutting-edge" technology. Finally, an open system strategy, like many other solutions, isn't a silver bullet that solves all problems a program manager may face.

Many of the issues that arise are in the detailed technical arena. While the program manager might be somewhat distant from such hands-on work – the contractor has the responsibility for delivering the system and the integrator for performing the labor of constructing the system – the program manager must be, at the least, aware of the changed nature of the tasks that

are to be performed. Without such awareness, the program manager is at a distinct disadvantage when unforeseen delays arise. Thus, we have included a section of issues faced by the contractor and integrator with which the program manager must also be familiar.

The guidance contained herewith doesn't claim to answer all questions that might arise. Its purpose is to provide an overview; above all, it does not give the program manager simple solutions to all problems. (In fact, we hope to convince the program manager that complex problems often require complex solutions!) The guidance does, though, provide a reasonably accurate picture of the realities that accompany extensive use of commercial items and an open system strategy.

The guidance contained herewith is most pertinent for the acquisition of commercially based software systems. To some extent it may also be applied to hardware and also to items that are a combination of both hardware and software. Although we recognize that hardware and combined items raise similar issues, the degree to which these issues affect the program manager is reduced. One reason is that hardware changes less frequently than software; thus issues of maintenance and upgrade aren't generally forced by changes in the commercial items. Perhaps the only area where there is as much change in hardware as in software is that of desktop computers where processing power, if not architecture, changes with an unprecedented rapidity. Fortunately, for the most part, the increase in speed isn't accompanied by change in functionality. This leads to the ability to simply replace older items with newer ones. (Though, it should be noted that, on rare occasions, for systems where real-time performance is a concern the increase in speed may be as harmful as a reduction.)

2 Using Commercial Items: Some Realities

Making extensive use of commercial items implies far more than simply purchasing products. Above all, it implies first, that we conceive, acquire, and sustain our systems with an understanding of the commercial marketplace, and second, that we are willing to accept the market's imperatives. This means that system acquisition is very different than it has been in the past. Particular differences are presented below.

A commercial-based acquisition relies on a different paradigm of system development.

The traditional way a system is developed starts with requirements, and proceeds through choosing an architecture, implementing the system, deploying the system, then testing and maintenance. In a commercially based system this paradigm is radically changed. Some (though not all!) of the task of "implementing" has been done by the commercial vendors; as often as not, their products have been created before the system is specified. And they are created not according to the system's requirements, but according to what their vendors believe will succeed in the marketplace.

The net result is that a simple sequence of development activities (whether the "waterfall" sequence or any other sequential model) no longer holds true. In its place, we see that system development depends on three activities – defining requirements, choosing an architecture, and performing market research – that occur simultaneously and continuously. Each activity constrains and informs other activities; they all affect one another. The nature of system development is thus cyclical, but not in the sense of "spiral" development.[1] Instead, the cycles we see in developing commercial-based systems are simultaneous; they permeate each other. We illustrate the difference between these two paradigms in Figure 1.

[1] Extensive information and recommended guidelines concerning spiral development are available from both academic and commercial sources. One source of this information is the SEI web site (http://www.sei.cmu.edu/cbs/spiral2000).

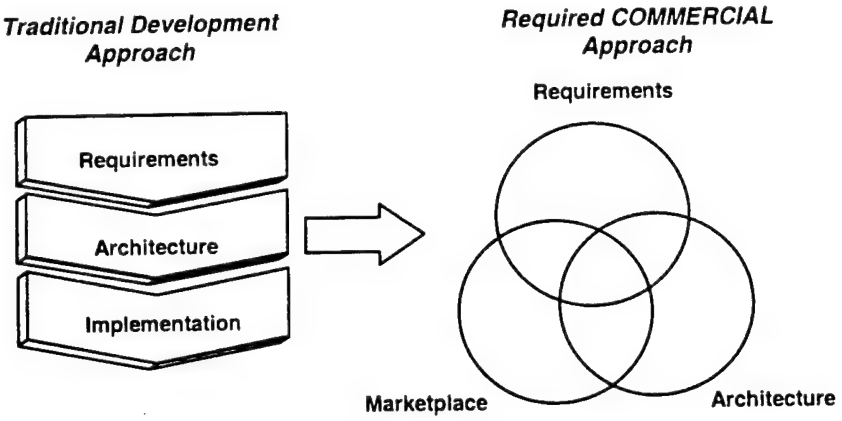


Figure 1: Traditional and Commercial Approaches to System Development

This new paradigm implies other realities as well. For instance

The difference between development through integration and development through coding is fundamental.

To the extent that a system uses commercial (or at least pre-existing) items as the basis of its functionality, it is inherent that such a system is integrated rather than written. It is increasingly likely that future systems will be a hybrid of COTS products, commercial items, non-developmental items, and custom code. The following table summarizes a few of the differences between the coding-based and integration-based approaches.

Development through coding	Development through integration
Design follows requirements	Requirements influenced by commercial items
Schedules driven by cost and staffing	In addition to cost and staffing, schedules depend on commercial vendors' schedules
Architecture is chosen by designers	Architecture is constrained by commercial items
Prototyping is desirable (but optional)	Prototyping is imperative

These differences demand that the contractor use a skillset that's changed from that used in the past. It is equally true that the job of the program manager, who has oversight over the creation of such a system, will also change.

Finally, the new paradigm means that

Using commercial items extensively means that many acquisition activities become continuous.

We stress the *continuous* nature of the activities in commercial-based system development: this characteristic pervades the entire acquisition process. As one prime example, we see this continuous nature manifested in the area of evaluation: assessment of commercial items is necessarily ongoing, since decisions based on one release will not necessarily hold true for succeeding versions of the same product. Thus, knowledge of commercial items is a decaying resource that must be continuously kept up to date.

More generally, the notion that certain activities have a localized place in the life cycle will now change; the different phases of the traditional life cycle (e.g., development, and maintenance) are blurred. In one sense, maintenance starts immediately. For example, during the development phase of even a relatively small system, there may be multiple releases of the commercial items it uses. It may even occur that a new trend can call into question fundamental technical decisions about the system (e.g., the rapid proliferation of HTML browser and web technologies forced many ongoing programs to re-evaluate networking and GUI strategies). This requires that developers continually make upgrade decisions regarding the developing system.

Likewise, the maintainer of a commercial-based system is often forced into decisions commonly regarded as development (e.g., comparing and choosing from among alternate commercial items and associated designs, making component buy vs. build decisions). This blurring of the life-cycle phases occurs because a commercial-based system is, in a sense, never completed, but continuously evolves through new component releases.

3 Commercially Based Systems: Guidance

The realities discussed above lead to new and possibly unexpected problems for a program manager responsible for the acquisition of a commercially based system. Thus, we suggest some of the issues that must be addressed during the life of a program. In each case, we also suggest courses of action that will minimize the effects of these problems.

The gap between the system requirements and available off-the-shelf products may be large.

Traditional government acquisition practice has been to define requirements in great detail and then have a contractor responsible for providing a system that conforms to those requirements. However, such an approach is unacceptable when acquiring a commercially based system. Typically, the gap between the capabilities of commercial items and the system needs will be large and dealing with this gap forms a significant part of the program manager's task. The program manager must manage the user's expectations. In particular, the users must understand that they are unlikely to be provided with an ideal solution, as it is unlikely that commercial items will fulfil all of their needs; instead, a "good enough" solution must be considered.

For instance, most commercial items embody specific process assumptions. These assumptions require that users follow certain business practices. If these business practices are different from a user's current practices then the user must adapt to the commercial items' practices. However, any change in daily work or operating procedures, even if it is for the better, can lead to the perception that the commercial approach is poor or unacceptable.

A program manager faces a constant temptation to modify the commercial items to reduce the gap and better fill the user's expectations. However, modifying the items may not be the best way to bridge the gap. Modifying the items usually leads to reduced commitment to the modified version from the vendor. The vendor may be prepared to make modifications on a one-time basis but, as the modified product isn't part of the vendor's product suite, upgrades to the modified version may be slow to arrive or non-existent. If the integrator makes the changes, then it is likely that increased costs will be incurred. Each time the vendor releases a new version of their product, the integrator must re-apply the modifications.

Suggestions

– *for managing user expectations*

- Use prototypes both early and frequently. Prototypes help users gain early insight into the new business practices required by the commercial items.
- Determine the list of all stakeholders (generally a wider group than expected – e.g., test and maintenance personnel should be included) and involve these stakeholders as early as possible in the life cycle.
- Educate users and decision-makers so that they understand that a commercial-based system acquisition requires frequent re-evaluation of requirements and frequent tradeoff decisions.
- Educate testers on the different needs of testing (e.g., how to test commercial items and how to determine if a commercial item is suitable)[Hissam 97].

– *for matching process assumptions between users and products*

- Determine which requirements are absolute and must be fulfilled exactly as specified and which are really user preferences. Prioritize the preferences and be prepared to trade away low-priority preferences that eliminate too many commercial items. One way to do this is through a risk-based strategy such as Win-Win [Boehm 94].
- Keep all requirements documents up to date; document all tradeoffs made during product evaluations.
- Conduct market research (and product evaluations) independently of the contractor.
- Participate in relevant conferences, trade shows, and user, professional, and standards groups.

Evaluation has a different role in commercially based systems.

The entire process of evaluation – from market research through gathering data to making tradeoffs and reaching decisions – is very different when a system relies on commercial items. Assessing different commercial items means comparing things that usually don't compare very well. What we really mean by "evaluation" in the context of a commercial product is determining its fitness for use in the context in which one hopes to use it. This is a very different concept from "test and evaluation" (where the word typically refers to "acceptance").

It is also important to evaluate *all* aspects of a commercial item. This includes not only characteristics of the item (e.g., security, information assurance, interoperability, reliability, and maintainability) but also characteristics of the vendor (e.g., the match between the vendor's support strategy and agency needs) and also the commercial marketplace (e.g., current technology and projected directions).

For assessing commercial items, there are many possible approaches. Examples include the Analytical Hierarchy Process (widely used in business decision-making)[AHP 00], technology delta techniques, model problems, risk/misfit comparisons, and feature checklists. Some of these are essentially oriented towards strategy, as in making a financial or a business case for choosing some particular product. Others are primarily technical, as in testing that a prod-

uct conforms to some collection of protocols. The point is that there may be many different approaches, and that following one approach can yield different results than following another. In practice, most substantial evaluations make use of multiple techniques.

Evaluation is the mechanism for gathering product information necessary for making system-engineering decisions. The decision-making process changes since the products, not the system requirements, will often drive the decision. This is an unfamiliar posture for many program managers (and their users), especially if such tradeoffs must be made repeatedly. It is equally unfamiliar for many contractors, who are more comfortable in simply meeting a defined set of requirements. However, the use of commercial items demands new behavior, and following the policy for extensive use of commercial items means that both the program manager, the user, and the contractor will consciously and explicitly accept this reality.

Suggestions

– *for understanding what "evaluation" means in the context of commercial items*

- Be familiar with different evaluation techniques – employ different techniques at different times in order to satisfy different needs.
- Verify the appropriateness of the contractor's plans for commercial item evaluation.
- Inform the contractor that you expect to be shown the rationale for making the choice for any given commercial item.
- Work with the contractor to select evaluation techniques based on the risks of making a poor selection.

– *for making commercial-specific tradeoffs*

- Require that the contractor specify which alternatives were considered.
- Determine in advance of contract award the strategic factors that may imply one product or another. This means understanding the business case for one solution over another.
- Employ testbeds for evaluating commercial items in the agency context.

Both program managers and contractors need a broadened skillset.

The construction of systems based on commercial items requires new technical skills of the contractor. The program manager and, indeed, the entire program office are affected no less profoundly. Starting with the program's acquisition strategy, all familiar notions about schedules, cost curves, or prototyping must now take into account the independent schedules and imperatives of the marketplace and the products in it. Program management style must also accept the inherent instability of the commercial world and be flexible enough to make (perhaps unfamiliar) tradeoff decisions with the contractor when commercial items and requirements do not match.

Even the way a program manager selects a contractor can be radically different. A company's business relationships and its vendor alliances are now as significant as having a large cadre of skilled programmers (though programmers are still important!). A company's ability to

make rapid accommodation to sudden technological changes – witness the speed with which Java has become a dominant technology – may be as significant as the traditional factors currently considered in source selection.

Perhaps the most intangible skill for the program manager is understanding the kinds of risks inherent in a commercial approach. Some of these risks stem from the lack of control that the program has over much of its software. Some arise because the fielded system will likely vary from the original requirements. Some risks occur because (at present) the ability to do complex commercial item integration tends to be less mature and has a smaller experience base throughout the industry. And some risks are familiar ones from traditional system development that do not disappear simply because the system makes use of commercial items. This last point is critical: the use of commercial items may minimize some risks, but does not remove them all, and no matter how much of a system is provided by its commercial items, the system still must be engineered. The program manager is ultimately responsible for ensuring that this takes place.

Suggestions

– *for planning cost and schedule*

- Make budget and schedule plans that account for the time and expense of learning unfamiliar commercial integration technologies (e.g., CORBA and COM).
- Don't use previous development-based acquisition plans as a template for a commercial-based acquisition.
- Examine the acquisition strategy to see where it can be made more flexible or better suited to the commercial aspects of the system in question.
- As part of market research, discover as much as possible regarding projected product releases and relate this to system schedules.

– *for choosing a contractor*

- Determine whether a commercial approach is employed in the contractor's internal systems.
- Determine whether the contractor has license management capabilities. Licenses and their management are very important when following a commercial approach.
- Determine the integrator's familiarity with third-party middleware technologies. Such familiarity is an indication of ability to integrate heterogeneous components.

– *for dealing with commercial item-specific risks*

- Make commercially oriented contingency plans and develop alternate approaches.
- Choose which aspects of the system to prototype based on commercial item-related risks.
- When considering any risk, investigate whether the marketplace complicates the risk.
- Be realistic about risk mitigation strategies; they may only shift and not mitigate the risk. For example, requiring commercial item modifications reduces the risk of rejection by users but increases the maintenance risks.

The ability to inspect the code of a commercial item is limited.

Typically, a major part of the job of the integrator is to inspect the code of components as part of a testing, verification, and certification process. However, when those components are commercial software the ability to perform such inspections is, at best, severely limited and, at worst, non-existent. Indeed, the more a system tends toward the commercial-intensive end of the spectrum, the more the development activity focuses on the role of putting pieces together. In spite of the hype about "plug 'n play," integrating commercial items requires expertise and detailed knowledge about operating system interfaces, communication protocols, and so on. For many systems today, particularly for information systems, this integration often entails working with third-party middleware (e.g., COM and CORBA).

The nature of the system may make different demands of the integrator. A system composed of a loosely coupled collection of distributed elements is likely to be easier to debug or upgrade than one where the elements are tightly coupled. The task of determining functionality, debugging, and repairing "black boxes" is very different from the various types of "debugging" possible when the developer has the source code. The fact that some components are commercial doesn't alleviate the need for system repair. Hissam provides technical discussions of this point [Hissam 97].

Suggestions

– *for understanding the contractor's technical plans*

- Understand the technologies used to integrate the commercial items. Discover what success the integrator has had with these technologies on previous programs.
- Determine whether components are to be loosely or tightly coupled and schedule accordingly.
- Have the contractor identify the system capabilities that are the focus of each prototype.
- Discover the integrator's testing and debugging strategy and the plans to deal with bugs in the commercial items.
- Determine how the contractor will develop expertise in specific commercial items.

Ensure that a strong vendor relationship is maintained.

During development, the integrator must, in addition to resolving technical issues, also perform tasks in the business arena. It will often be the case that the government represents only a small fraction of a vendor's business. In such cases, the integrator will have little "leverage" over the business practices of the vendor. Thus, the integrator must be experienced in vendor negotiation and license management. The integrator will also depend upon strong relationships with the vendors for practical reasons. Vendor agreements can include commitments to make unscheduled releases or to make patches available that fix important bugs. Without a strong relationship, the integrator may make development decisions based on limited or inaccurate information.

It is also important for the program manager to maintain a strong relationship with the vendors. The integrator's task is over when development is complete and the responsibility for system maintenance then falls on the program manager (though this may be contracted out – even to the integrator). The maintenance role, with respect to commercial items used within the system, is much the same as that of the integration role – to incorporate the appropriate versions of the commercial items into the system. A strong relationship with the vendor will lead to decision-making based on sound information.

The risk, for both the program manager and the integrator of using faulty or incomplete information is that decisions may be optimistic with respect to the vendors' release schedules or functionality available in future releases. Dependence on a commercial item available in the future leads to the risk of a delay in the item's release schedule leading to slippage in the system's schedule.

Suggestions

– *for dealing with schedule slippage*

- Identify schedules, releases, and capabilities that are dependent on future commercial releases.
- Understand the contingency plans for delays in commercial releases.
- Identify whether negotiations with vendors concerning release schedules have occurred and determine the vendor's track record of keeping to schedule.

Integrating with legacy systems requires architectural tradeoffs.

It is a very rare system that operates wholly independently. Each system has interfaces with others whether existing (legacy) or under construction. Commonly, the interfaces to legacy systems were defined in the past with little, or no, knowledge about future technology directions. Thus the interfaces, while ideal for their day, may no longer be best suited for a modern system. For example, a legacy system may distribute data among the components using a shared memory model while a modern system would use middleware such as CORBA. In order for the systems to be integrated, this communication mismatch must be resolved.

The program manager is faced with a number of choices, each of which has costs and benefits:

- Modify the architecture and interfaces of the new system to adapt to the legacy system's model. This approach has the benefit that the new system can be designed to fit the legacy system. However, there are costs: the architecture may be less than ideal; commercial items may not be available that can interface to the legacy system; and new technologies are ignored in favor of propagating old technologies.
- Modify the legacy system interfaces to accommodate the new system. This approach has the benefits of introducing new technologies and providing the system designers the opportunity to create an optimal architecture and design. The cost, though, is that the code for the legacy system may no longer be modifiable (e.g., shortage of devel-

opers with the appropriate language skills, failings in the development environment, and, in the worst case, the inability to rebuild the legacy system). A further cost is that such modifications may compromise the legacy system architecture.

- Create adapters that perform the communication between the new and the legacy systems. The benefits are that the minimal disruption is caused to both the legacy system and the design of the new. The cost is that using adapters is rarely successful as a long-term strategy. Adapters are, by their nature, tied to the interfaces of both systems and therefore tend to be brittle and to resist changes. Furthermore, use of adapters may overburden heavily used interfaces.

A program manager may not be able to choose the optimal strategy for a given system but must make the choice in the context of the other systems through negotiation with not only those systems' managers but also their installed user base.

Suggestions

– *for choosing a legacy system integration strategy*

- Consult the managers of the legacy systems to determine which strategy is feasible. If the legacy system is to be changed, negotiate a budget and schedule.
- Use a testbed, if possible, to ensure validity of the integration using typical data.
- Keep a record of all interface assumptions where they impact commercial items in the new system – these will form the basis for re-integration following commercial item upgrade.

Upgrading commercial items is a reality that must be faced.

One of the benefits of using commercial items in a system is that the vendor performs the maintenance, the costs of which are amortized across all purchasers of the item. However, this benefit brings with it the cost of installing an upgrade of an item back into the system. This cost is made more significant by the fact that upgrades will appear according to the vendor's and not the program manager's schedule. Further, not installing the upgrades may lead to the situation where the system's version of a commercial item is so old that a support contract no longer covers it.

A significant risk is the withdrawal of vendor support for a commercial item. Vendors will sign support contracts, but these are typically of limited duration; for long-lived systems it is likely that the system will outlive such a support contract. In the extreme case, a vendor may simply go out of business or be acquired by another company with no commitment to supporting the items in use. Although the source code may be escrowed, accepting responsibility for subsequent maintenance will affect the maintenance budget for the system. In the worst case the program manager's new maintainers will be unable to perform maintenance either due to shortage of personnel, lack of expertise, or lack of an appropriate support environment.

Given that upgrades are a reality and that the integrator may have no insight into the inner workings of the commercial items included in the system, a modified certification strategy must be used – it must be as much an investigation as a certification. Certainly, the vendor's release notes will accompany each upgrade; however, such release notes are frequently terse and only cover the

major changes. Upgrade testing requires not only certification that the item still does what the system needs, but also investigation to ensure that no new features have been added that make the item unsuitable for use.

Suggestions

– *to manage change*

- Base interfaces on publicly recognized standards that are widely accepted in the marketplace.
- Monitor the marketplace for technology enhancements.
- Maintain a close relationship with vendors in order to gain insight into new releases.
- Plan for making maintenance upgrades even during development.
- Assess each upgrade to determine whether or not it can be skipped without causing system problems.
- Participate in the integrator's integration-and-test strategy.
- Establish plans to work with vendors for timely problem resolution.

– *to test commercial items*

- Unless it is impractical, use testbeds to certify commercial item upgrades. Particular attention should be paid to unanticipated side effects in critical areas (e.g., safety, security, and performance).

Multiple commercial items compound the problems.

All of the problems previously discussed occur when only one commercial item is incorporated into a system. When there are multiple commercial items, the problems are exacerbated due to the dependencies that one item may have on another.

System stability becomes an important issue – the average time between upgrades for commercial items is between 6 and 12 months. This means that, as the number of commercial items increases, the system may never achieve stability if every upgrade is immediately integrated. Typically, the upgrade frequency leads to multiple configurations being maintained in the field. This places a high burden on a configuration management system designed to maintain a single baseline. Some commercial items may depend on specific versions of others – this means upgrading one item may cause a chain reaction causing the update of one or more other items. Given that each vendor will perform upgrades on his or her own schedule, the configuration management system must be able to select among multiple versions of an item in order to deliver appropriate system configurations.

Suggestions

– *for managing change*

- Look for collections of inter-dependent commercial items. Plan to upgrade them all simultaneously. Prepare tests for each collection.
- Ensure that rigorous configuration management is exercised.

4 Using Open Systems: Some Realities

The phrase "open system"[Meyers 96] calls to mind a vision of a system that is flexible and "open" to the use of many products from different vendors. The phrase often conveys the image of a world where there is an easy "plug 'n play" between components that were not originally designed to work together, but where the products both conform to the same open system standard. A second image that arises is one where the system can take advantage of the latest offerings from the commercial marketplace because components adhering to the standard are interchangeable.

As stated in the definition, an open system strategy relies on specifications that are fully defined, available to the public, and maintained according to group consensus. These requirements seem to call for standards, yet there are typically many debates over what is and what is not a standard. We will sidestep this discussion by using a broad definition of standard, i.e., one that includes formally approved standards (e.g., those from ISO/IEC), standards defined by consortia (e.g., those from the Object Management Group (OMG)), and *de facto* standards (e.g., the Microsoft Windows interface). The common thread among these specifications is their acceptance: either in the formal sense that they have an accepted pedigree or in the sense that the marketplace informally agrees to adhere to those standards as can be seen by the number of people buying conformant products and the number of products that adhere to the specification. It is only with this latter view that we can consider standards such as Microsoft's Windows interface or Sun's Java specification as being standards.

The standards that support open systems will most often be in the form of interface standards where the application program interfaces (APIs), data formats, or protocols are defined. This is not always the case. The Defense Information Infrastructure Common Operating Environment (DII/COE), for example, standardizes on both products and the ways in which the products are described rather than on interfaces as such (though DII/COE does call out conformance to various interface standards for their levels of compliance).

Interface specifications are generally insufficient to ensure full "plug 'n play."

In practice, the real interface between two components of a system is more than adherence to an interface specification; it also includes all of the assumptions that each component makes about the other. The APIs, data formats, and protocols comprise a significant number of the assumptions, but certainly not all – sequences of API calls, or the meaning the application ascribes to the data items in the defined formats go beyond most interface specifications. Instead of "plug 'n play," the reality is that a significant amount of effort is usually needed to

make two independently developed components inter-operate, even when they conform to an interface standard.

Another issue faced by interface standards is that of interpretation. Since standards are written, sometimes intentionally, in ambiguous language, different component developers may interpret the standard in different ways. Another source of ambiguity arises from options in the standards. Options allow different developers to implement the standards in different ways and can lead to components that do not inter-operate.

Standards development and evolution is a slow process.

With the possible exception of the *de facto* standards, the development of a standard where group consensus defines the content is a slow process. The greater the desire for consensus, the longer it takes to develop the standards. Some formal standards have taken many years before becoming accepted because it is difficult to achieve consensus in a forum where the individuals have different vested interests (e.g., the competitive advantage of their employers). The result is that formal standards generally employ older rather than cutting-edge technology. Even the OMG standards which, initially, were developed very rapidly, have slowed in pace. This is a natural consequence of the acceptance of the standard – as more products conform to the standard, resistance to change increases because each product vendor has a vested interest in keeping the standard as is.

De facto standards are the most rapidly changing, in part because a small group with a common interest generally defines them. Witness, for example, the number of instances of the Java standard. In each case, the updated standard has been an improvement on earlier versions and has been a response to user needs. However, the standard is, essentially, proprietary and wholly under the control of Sun Microsystems.

Standards generally define the lowest common denominator.

Standards are typically developed by groups of experts in the subject area coming together and agreeing on the common elements of the subject area. Unfortunately, those experts all arrive with (even if subconscious) vested interests – perhaps the health of their company or their own views of what really defines the subject area. Thus, initially, the experts may disagree about the subject area more than they agree. Even exhaustive discussion is unlikely to achieve total agreement among the experts as each works to protect individual interests. The consequence is that the resultant standard, which defines the areas of agreement, is a minimal rather than a maximal definition of the subject area. Many useful parts of the domain are omitted or, at best, are only present as options.

Standards conformance means different things to different people.

It is frequently possible for components to conform to standards in different ways. Unless the nature of conformance is clearly spelled out, this will frequently mean that even components that conform to the standard may not inter-operate. For example, DII/COE spells out eight levels of conformance in order to define the degree of inter-operation.

One concept of importance is between the notion of "conforming" and "strictly conforming." A component that "conforms" to a standard may provide an implementation of all the features defined within the standard but, in order to gain market advantage, the vendor may add additional features in order to make the product more attractive to the consumers. Certainly, the product conforms (after all, it implements all of the standard), but the unwary consumer who takes advantage of the additional features is, once again, stepping onto proprietary ground and may lose the ability to plug in a different product conforming to the standard. A "strictly conforming" component is one that implements the standard and no more. Coupled with this is the notion of a "strictly conforming application"–an application that uses no feature outside of, or data value above–the minimums defined in the standard. Unfortunately, as we've already remarked, the standards are frequently the lowest common denominator and a strictly conforming component or application may not be sufficient to meet a system's needs.

5 Open Systems: Guidance

As for commercial-based systems, we can offer the program manager some guidance in avoiding the problems associated with open systems. However, this guidance cannot be given without considering commercial items. Thus we will present the guidance relating to both commercial items and open systems within this section.

The variety of competing standards makes the selection among them more difficult.

One of the disadvantages of standards is that there are many, sometimes competing, standards to choose among. More importantly, unless a system is trivial, no single standard will be sufficient for the system's needs. The selection of appropriate standards is an important choice that can be made by the program manager, the contractor, or both. In either case, however, the program manager should have confidence that the selection is harmonious and that a system composed of components conforming to those standards is both achievable and desirable.

Each standard that is selected affects the overall system architecture. Over-specifying the standards will lead to systems with sub-optimal designs. Under-specifying the standards will lead to systems that don't inter-operate with other systems.

The nature of the standard affects the risks to the system. Widely supported formal standards, because of their nature will,[2] if they suit the system's needs, be more likely to be of long term value than *de facto* standards. On the other hand, the latter may be more easily influenced and, if changes can be suggested and incorporated into the standard, the system will benefit. The risk from *de facto* standards, though, is that their mercurial nature may leave the system dependent on features of an out-of-date standard. Or, changes introduced by other groups may adversely affect the system. For example, we have already seen a number of commonly used HTML tags deprecated in the latest version of that standard. Because many different vendors participate in more formal standards development, there is an increased likelihood of commercial items being available for use by the system.

It is important to consider the age and future of a standard when making the choice to use it as part of a long-lived system's interfaces. If the standard is near the end of it's life it may be dropped too soon for it to have been a worthwhile investment. Similarly, at the very early stage of its life, a standard may be untested and open to significant change. An indicator of the maturity of a standard is the presence of commercial items supporting that standard. If

[2] A standard is widely supported when there are multiple implementations provided by vendors. A formal standard that is not widely supported may accrue no long-term benefits to the system.

there are few such products, and the standard is new, then careful consideration should be made before adopting it as part of the strategy. For example, consider the OMG; there have been many, significant changes between the three major versions of the CORBA standard.

The standards chosen for a system require maintenance just as much as the system itself. As standards evolve, the program manager in conjunction with the contractor and stakeholders of other systems using the standard must determine when it is appropriate to migrate toward conformance of an updated version of the standard. Ideally, the new standard will be compatible with the existing one; but if not, the system may support multiple similar standards until all interacting systems can be upgraded appropriately. This places additional burdens on the configuration management system, as it must be used to manage the versions of the standards as well as the components.

We have stated that most standards define interfaces between products and do not mandate particular instances of commercial items. The advantage of such a strategy is to be seen in maintenance; there is no need to change the standard simply because a new version of a commercial item is available. If the standard defines specific versions of commercial items then, if it is not to become out of date, it may need to be updated each time an upgrade is available. At the least, the new version should be investigated to determine whether it should be incorporated into the standard. If too many items are incorporated into the standard, then the effort involved in ensuring compatibility of the items will increase significantly, perhaps to the stage where it isn't possible for the standard to keep pace with the marketplace. One advantage of a product-based strategy is that the maintainers of the standard will ensure some level of compatibility between products where an interface-based strategy leaves that to the users.

Suggestions

– *for choosing appropriate standards*

- Form working groups with managers of inter-operating systems to determine the appropriate interface standards.
- Develop a model of the role the standards play in the overall system.
- Clearly state which standards are mandatory – any that interface to external systems should be in this category.
- State your preferences for standards – prefer formal standards, but accept *de facto* standards when appropriate.
- Choose commercial-based rather than government-based standards, where possible.
- Understand the contractor's justification for the selection of each standard.

– *for maintaining standards*

- Participate in standards organizations so that the system's interests are represented.
- Expect and plan to change some (or all) of the standards during the life of the system.

Develop expertise in conformance management.

It should be noted that conformance, regardless of the nature of that conformance, means that the component behaves according to the standard. This doesn't guarantee that it will interoperate with other components in that the standard cannot address implementation assumptions. Nor does it imply anything about attributes of the implementation (vendor extensions) that are not specified by the interface standard.

The way you determine conformance to a standard is through testing. The nature of conformance testing varies depending on whether the component being tested is an implementation of a standard or an application that uses the standard to interface to some other component (or an external system).

Testing implementations for conformance may be as simple as applying a test suite ensuring that each function of an API (or packet of information in the case of protocols) behaves according to the standard. This test suite must be capable of being repeatedly applied without need for human intervention. (The latter causes randomness that will reduce confidence in the conformance testing.) However, such testing only determines whether the implementation does what it is supposed to do, and does not determine whether it provides additional capabilities that may be of value to the system. These additional capabilities should be used with caution, as it is unlikely that other vendors will make the same extensions; however, the benefits of using the extensions can often outweigh the risk of increased cost if the implementation has to be replaced.

Testing applications for conformance is a harder task, particularly if the application is required to be strictly conforming. The easiest way to perform such testing is through automatic inspection of the source code. Of course, for commercial items source code may not be available and investigative testing as discussed by Hissam may be used.

Many standards are of a general nature and define many features, not all of which are pertinent to the needs of a system. A profile is a set of one or more standards and, where appropriate, an identification of chosen subsets, options, and parameters of those standards necessary for accomplishing a particular function. Investigation into the applicability of existing profiles or the development, if necessary, of new profiles, will reduce the overall cost of conformance management. Use of profiles that subset existing standards are also likely to increase the number of commercial items available for use by the system.

Suggestions

– *for determining conformance*

- Spend most effort determining conformance for aspects of the standard pertinent to your system.
- Use existing profiles if possible; if not, develop them where necessary.

- Avoid vendor extensions when possible and, if extensions are used, document all that are used.
- Maintain a database of the standards specified for the system and the conforming commercial items.

Expect changes in schedules and costs.

The budget and schedules for an open system strategy are different from those of a traditional development, even one conforming to known military standards. In particular, the program manager should expect that initial costs will increase and that long term costs will decrease. Typically, schedules are closely related to costs – as the latter increases, so the former lengthens. Thus the following discussion will focus on cost and assume the relationship to schedule.

The increase in cost arises from the need to perform the selection of standards (and the development of any necessary profiles of those standards). Coupled with the selection of standards is a market survey to determine whether any conforming commercial items are suitable for use. Indeed, adopting an open systems strategy may also increase cost due to the need to train personnel (both in the Program Office as well as for the contractor) in effective use of the strategy.

Another cost that increases is that of the testing strategy. As the conforming components are upgraded, it is necessary to ensure that the upgrades still conform to the standard. Divergences, particularly in the case of features used by the system, need to be noted and understood. A positive aspect of this conformance certification is that it may be performed with the necessary re-certification of commercial item upgrades for suitability to the system's needs.

Long term (and even some development) costs should decrease because of the expectation that commercial items will already exist that conform to the chosen standards. In such cases, it is generally cheaper to purchase the commercial items rather than develop agency-specific versions of those items. Additionally, if there are conforming commercial items, competition between vendors will tend to drive the purchase price downwards. Also, replacement parts (or support) may be available from third parties again at potentially reduced cost. Finally, the vendor or some other independent testing laboratory may have already performed some of the conformance testing. This reduces and may even eliminate much of the need for re-testing by the integrator.

Costs and schedules will also vary according to the contractor's experience with respect to open systems. A contractor with a background of using open systems should already have the necessary skills to adopt an open systems strategy. Consequently, costs should be decreased and schedules shortened as the contractor will not need to spend time learning about open systems.

One important change to the schedule is that architectural decisions will be made earlier in development and will depend upon the standards and the available commercial items. Since

these architectural decisions are fundamental to the overall long term support and upgrade strategy, more attention to and awareness of the implications of these decisions is needed.

Suggestions

– *for selecting among contractors*

- Look for previous contractor experience with open systems in general and the chosen standards in particular. Ask for evidence of previous successes.

– *for planning budgets*

- Plan for additional systems engineering effort.
- Take into account the new costs – don't base the budget and schedule plans on a previous system that didn't use an open system strategy.
- Budget for a testing capability that can be continuously employed.

Understand the potential conflict between an open systems and a commercial-based system strategy.

An open systems strategy can be pursued independently of a commercial-based system strategy and *vice versa*. At times, conflicts will arise between the two strategies and one strategy must be chosen over the other.

Conflict arises when there are no commercial items that strictly conform to the standards. The commercial items may only implement a large but incomplete subset of a standard. For example, neither Netscape Navigator nor Internet Explorer is fully compliant with the current HTML standard for web pages. The result is that the replacement of one, broadly compliant, commercial item with another may give rise to crucial incompatibility problems, requiring at least a redesign of custom code or applications.

A second conflict arises when a commercial item has desirable features but does not conform to a particular standard. This typically arises when vendors add features to their products in order to increase market share. Again, if we consider Netscape Navigator and Internet Explorer, we see that they each embody extensions to the HTML standard. For competitive reasons the extensions are incompatible. In such cases, the choice must be made between creating strictly conforming applications (and thus ignoring possibly useful extensions) and using vendor extensions—but with the realization that replacing the component will require some redesign. It is unthinkable (particularly in our browser example) for a program to either develop a strictly conforming component when commercial components (e.g., the browser) exist, or even to wait for the vendors to make strictly conforming products available.

It is also important that the standards describe commercially viable systems. Even though standards evolve, they do not do so as frequently as the marketplace. Given the expected lifetime of the agency systems, it is likely that one vendor or another will drop support for a particular standard from their product suite. If the contractor has chosen commercial items (and

vendors) wisely, then the evolved versions of those items will still serve the agency needs, even if the standards no longer do so.

The program manager must then consider what purpose is served by adhering to the standard. If the purpose is to achieve inter-operability with other systems there is a choice of approaches; commission changes in the commercial items to support the standard or upgrade the interface between the systems. The latter approach is certainly preferable. The former approach brings with it the same risks as customizing a commercial item during initial development.

Suggestions

– *for choosing between strategies*

- Prefer products to standards, but look for products that implement all of a standard.
- Document all extensions to the standard that are used within the system – it will make replacement easier in the long term.
- Persist with standards for as long as possible, but be prepared for changes. If the commercial world drops a standard, consider why it is still being required.

References/Bibliography

[AHP 00] *Analytical Hierarchy Process* [online]. Available WWW<URL: http://www.cs.adfa.oz.au/teaching/studinfo/da2/lectures/L14html> (2000).

[Albert 00] Albert, C. & Morris, E.J. *Commercial Item Acquisition: – Considerations and Lessons Learned* [online] Available WWW<URL: http://web1.deskbook.osd.mil/reflib/DDOD/005EO/005EODOC.HTM#T2> (2000).

[Boehm 94] Boehm, B.; Bose, P.; Horowitz, E.; & Lee, M.J. *Software Requirements as Negotiated Win Conditions,* ICRE 1994. Available WWW <URL: http://sunset.usc.edu/TechRpts/Papers/NGPM-Requirements93.ps>

[Brownsword 99] Brownsword, L.; Oberndorf, P.; & Sledge, C. *COTS-Based Systems for Program Managers* (briefing). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. Available WWW <URL: http://www.sei.cmu.edu/cbs/coursedesc/CBSforPMsBriefDisc.html - size 20.0K

[FAR] Federal Acquisition Regulations{excerpts pertinent to commercial products. Available WWW< URL: http://www.dsp.dla.mil/documents/sd-2/chapter1/htm>

[Hissam 97] Hissam, S. *Case Study: Correcting System Failure in a COTS Information System.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997. Available WWW<URL: http://www.sei.cmu.edu/cbs/papers/monographs/case-study-correcting/case.study.correcting.htm>

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	April 2001	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Guidance on Commercial-Based and Open Systems for program managers	F19628-00-C-0003

6. AUTHOR(S)

Patrick R. H. Place

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2001-SR-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

With increasing frequency, program managers are acquiring systems based on both commercial products and open systems. This brings both benefits and risks. Although the benefits of using commercial products and open systems are widely publicized, the risks are often overlooked. This document discusses various risks and provides guidance that may be used to mitigate those risks.

14. SUBJECT TERMS	15. NUMBER OF PAGES
COTS development, COTS acquisition, commercial item strategy, open systems strategy, commercial-based systems, open systems	41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102